
OTX IN PRACTICE

# Barrier-free exchange of **test sequences**

Mastering complexity is a challenge of our time. This can only be accomplished through interdisciplinary collaboration. Integrated, standardised processes are a prerequisite. Standards establish transparency and exchangeability. They form the basis for continuous improvement, consistent service provision and avoiding redundancies.

The OTX (Open Test sequence eXchange) standard according to ISO 13209 is a domain-specific language (DSL) for the reliable description of exchangeable and executable testing logic in the automobile industry. Diagnostics sequences can be created graphically and simultaneously described in sufficient detail so the same testing logic can be executed in any of various target environments, without barriers. The standard is mature and comprehensive enough to replace existing solutions in development, production and the workshop. Possible uses for OTX range from describing simple functional tests in development to start-up procedures in production to fully generic tester applications with guided troubleshooting in customer service. OTX is open and stable as well as platform and technology-neutral.

## Exchangeability and conformity with standards

The standard alone is not sufficient to ensure that the promising potential is more than just advertising messages, but can actually be realised in practice. Processes have to be adapted, tools developed, and comprehensive thinking fostered. OTX alone does not ensure exchangeability. Here exchangeability me-

ans the use of unaltered OTX documents in various target systems. Put another way: The same testing logic has to be executable in a wide variety of systems and lead to the same functional results. In order to ensure this exchangeability, the test sequences (OTX documents) have to meet the following criteria:

- Validity
- Completeness
- Target system independence

Valid OTX data must use the correct syntax – corresponding to the data model – and not violate any critical semantic checker rules. This is not a problem in most cases. Mistakes that occur here tend to be gross errors that are quickly located.

Furthermore, a test sequence is complete when the entire testing logic exists in OTX. Problem: The definition of what the testing logic encompasses depends on the specific case. Opinions can vary widely. Ultimately, the question is what one wants to achieve with OTX. Two extreme cases illustrate this point: In the first, OTX merely calls a function in an external system that contains the entire testing logic. In the second case, the entire tester including the HMI, administration of roles etc. exists in OTX. The first case may be valid, but is not complete and therefore also not exchangeable. The second case is both valid and complete, but pushes the boundaries of OTX's expressiveness; OTX was not made for this. It has been shown in practice that the testing logic stored in OTX should approximately correspond to the technical knowledge of an adept person responsible for a component.

Once the testing logic is clearly delimited, completeness also includes that all elements referenced in OTX are present and accessible. For example, one could call a procedure that does not exist in OTX at all, but is somehow linked to the external method of a test standards library at runtime.

## Target system independence

OTX must not contain any target system dependent data. Otherwise it may be valid but is not exchangeable. Target system independence begins where specific environments with specific implementations are needed to execute the sequence. Since a sequence is always executed on a target system, this dependency exists in all cases. The configuration data required for this cannot not be within but must be stored outside the OTX documents; see OTX mapping below.

A place where target system dependent data can be stored is what are called metadata. Metadata can be stored on almost all elements in OTX, thereby transporting additional data such as release versions or procedure identifications at will within the OTX document. However, these data are not permitted to influence the runtime behaviour of the testing logic. A simple test: After erasing all metadata, the sequence still has to be executable and the runtime behaviour is not permitted to change.

The interface between OTX and the outside world (procedure parameters, context and status variables, screen parameters etc.) is another potential doorway for target system dependence. Only convertible data types such as Boolean, integer, float, string, bytefield, list, map, enumeration and structure may be used here.

In summary, it can be said that OTX data input is only truly compliant with the ISO 13209 standard if it is valid, complete and target system independent. All tools from the company EMOTIVE generate and process standard-compliant OTX exclusively. That being said, there certainly are meaningful cases using valid OTX that is incomplete and target system dependent. However, exchangeability across process boundaries with different tool landscapes is then no longer assured.

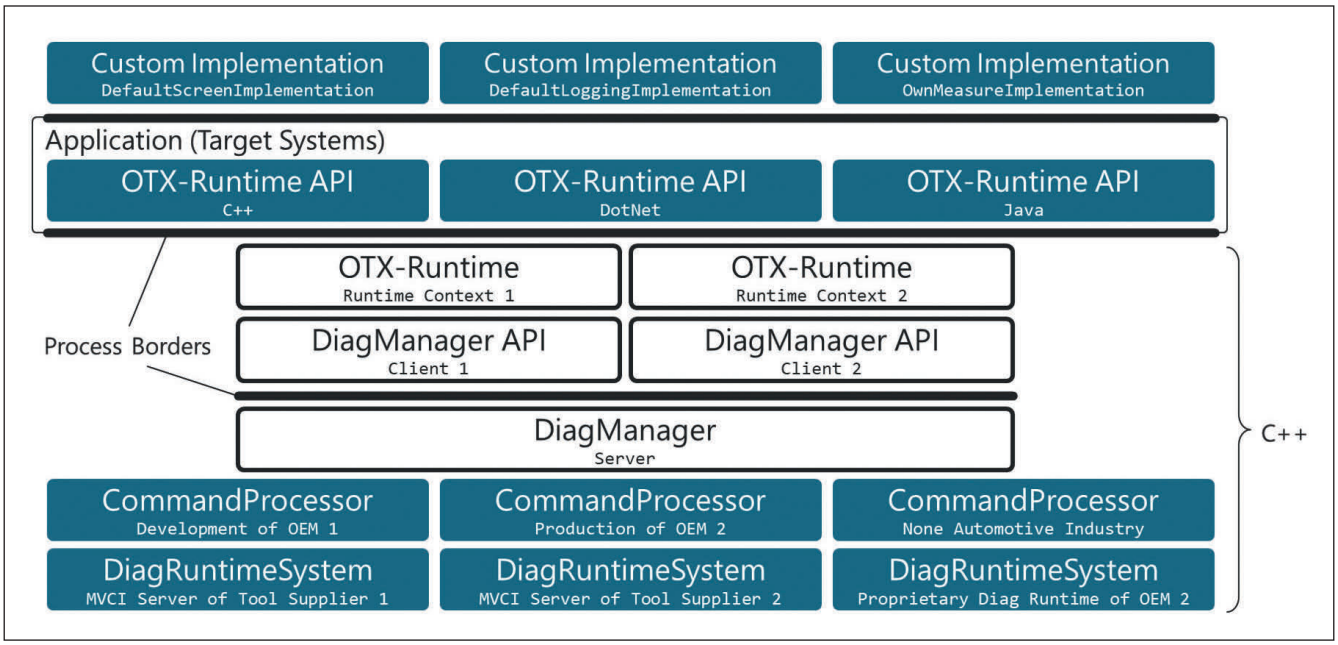


***Figure 1: Exchangeable levels of the EMOTIVE OTX Runtime.*** © emotive

## Exchangeability and tool integration

EMOTIVE has made major efforts in recent years to implement this standard in

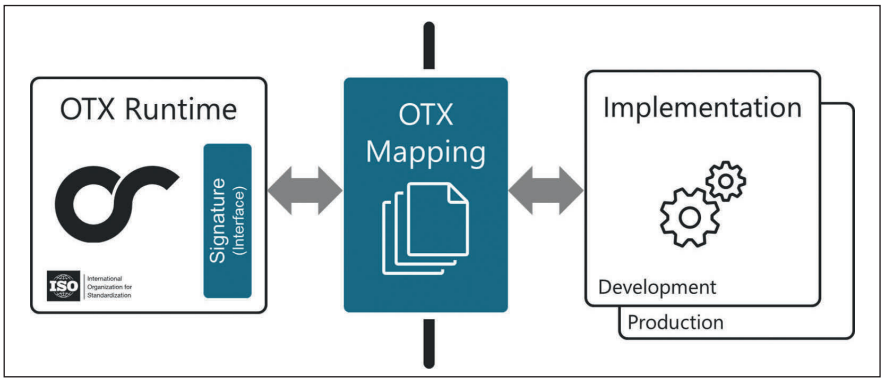


***Figure 2: Functional exchangeability (OTX mapping).*** © emotive

practice. A completely new runtime environment for OTX (OTX Runtime) is available, developed with the requirement of enabling exchangeability on all meaningful levels; see Figure 1.

To execute OTX, the OTX Runtime API has to be integrated into an application. OTX projects (PTX files) can be loaded via the API, their data structure determined (browsing) and procedures started. The OTX Runtime API is available in three technologies – C++, DotNet and Java – so it can be implemented within practically any existing technology. A runtime context is generated upon starting a procedure. It is the heart of the runtime environment. The actual execution of OTX takes place here. It as well as all underlying layers are written in native C++ for high performance and resource-saving in any target architecture, such as desktop, web or embedded.

## Diagnostics communication

The OTX Runtime also works on the available standalone Diag Manager for diagnostics communication. Its task is to translate all OTX commands relevant for diagnostics to the commands of a specific diagnostics runtime system. It can work as a server so that parallel diagnostics from discretionary processes or applications are possible. The server serialises the commands of the clients and passes them on to the Command Processor. The Command Processor optimises and prioritises the diagnostics commands. For example, the Command Processor administers the open diagnostics channels. The Command Processor is exchangeable. This allows users to implement their own, specific administration of the communication channels or diagnostics services. The Command Processor sends its commands to the Diag Runtime system. The actual translation from OTX to the specific methods of a diagnostics runtime system, for instance according to ISO 22900–3 (MVCI), takes place in the Diag Runtime system. This step is entirely exchangeable as well. Here users can independently connect to their own, proprietary diagnostics runtime systems.

But OTX does more than just diagnostics communication. OTX includes various extensions for interacting with other external systems:

- CommonDialogs
- Context and StateVariable
- ExternalServiceProvider
- HMI
- i18n
- Logging
- Measure
- SQL
- TestResultHandling

These extensions are not implemented within the OTX Runtime, but externally in what is called the custom implementation. Interfaces that users can implement themselves are provided for this purpose in the OTX Runtime API. EMOTIVE supplies standard implementations for these interfaces. It is therefore possible to seamlessly integrate and execute OTX in any target system. The same OTX sequence can be executed in a web application with an HTML Screen connection or in a vehicle infotainment system.

## Functional exchangeability

In order to execute OTX testing logic on a target system, external OTX calls have to be bound to specific functions in the target system. EMOTIVE calls this OTX mapping; see Figure 2. All mapping information required for a target system is stored in a file. The same OTX testing logic can run in various target environments, simply by exchanging this file.

In OTX mapping, screens (corresponding to a screen signature and its parameters in OTX) are, for example, bound to appropriate classes of a DotNet assembly that represents a window in WPF technology or, in another case, bound to an HTML template that describes a page in the Internet browser.

## Conclusion

Provided that compliance with the standard is consistently observed from creation to execution, OTX is unrivalled for exchanging quality proved testing knowledge across process and tool boundaries. OTX can ensure that the same unaltered testing logic is executable at all times in any target system and leads to the same results.

The new OTX runtime environment from EMOTIVE guarantees platform independent exchangeability on all meaningful levels. ■ (oe)

**www.emotive.de**

**Dr. Jörg Supke** is the CEO of emotive GmbH & Co. KG, 73760 Ostfildern.